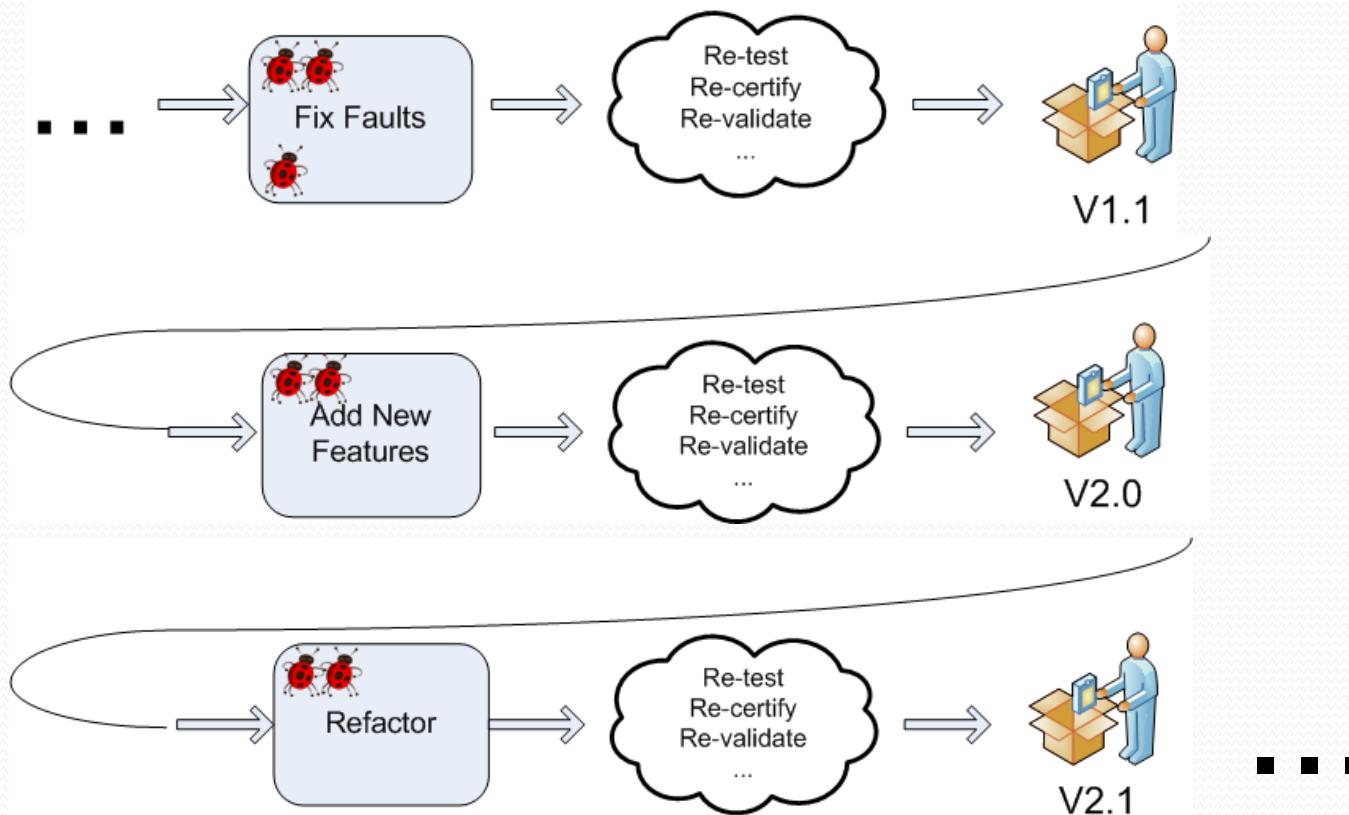


Differential Symbolic Execution

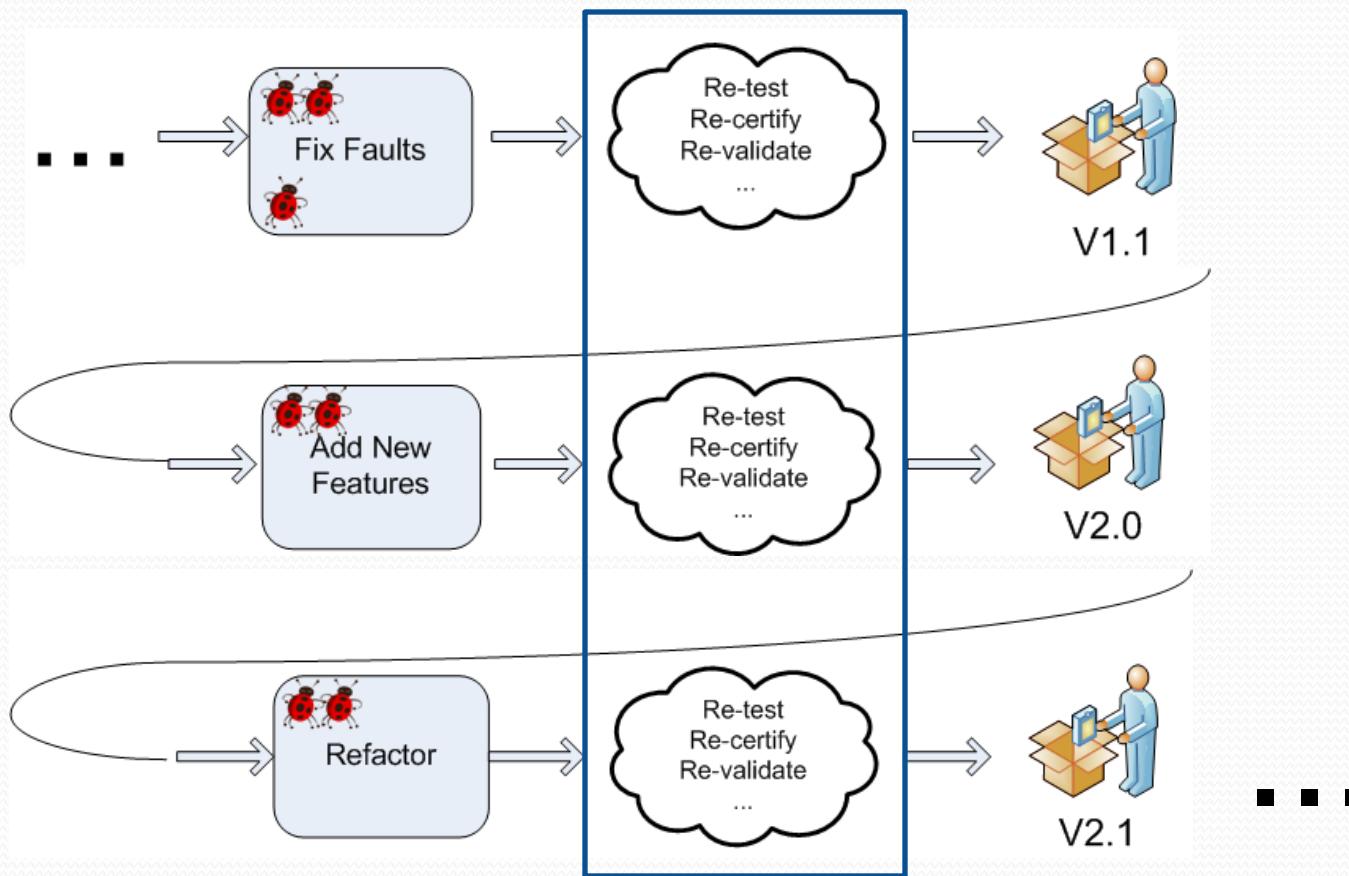
*Suzette Person
Dissertation Defense
July 8, 2009*

PhD Committee
Matthew Dwyer, Advisor
Myra Cohen
Sebastian Elbaum
David Rosenbaum

Software Evolves



Software Evolves



Motivation

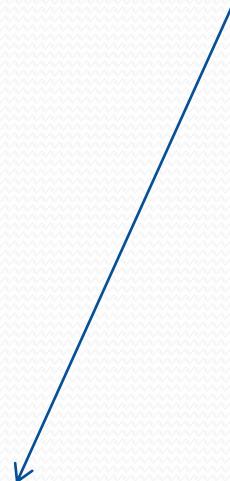
```
while(CONDITION)
    switch (CONDITION) {
        ...
        case ox5c /* '\\\' */:
            if (CONDITION) return T_UNKNOWN;
            switch (CONDITION{
                case ox76 /* 'v' */: tmp.append(oxob /* '\v' */); break;
                ...
            default:
                if (CONDITION && CONDITION) {
                    ...
                    if (CONDITION && CONDITION && CONDITION) {
                        ...
                        nextByte();
                        if (CONDITION && CONDITION && CONDITION) {
                            nb = (byte)(nb * 8 + currByte() - ox30);
                        }
                    }
                }
            ...
        }
    }
```

location

Motivation

```
while(CONDITION)
    switch (CONDITION) {
        ...
        case ox5c /* '\\' */:
            if (CONDITION) return T_UNKNOWN;
            switch (CONDITION{
                case ox76 /* 'v' */: tmp.append(oxob /* '\v' */); break;
                ...
            default:
                if (CONDITION && CONDITION) {
                    ...
                    if (CONDITION && CONDITION && CONDITION) {
                        ...
                        nextByte();
                        if (CONDITION && CONDITION && CONDITION) {
                            nb = (byte)(nb * 8 + currByte() - ox30);
                        }
                    }
                }
            ...
        }
    }
}
```

What are the effects??



Motivation

```
while(CONDITION)
switch (CONDITION) {
...
case ox5c /* '\\\\' */:
    if (CONDITION) return T_UNKNOWN;
    switch (CONDITION{
        case ox76 /* 'v' */:  tmp.append(oxob /* '\\v' */); break;
        ...
default:
    if (CONDITION && CONDITION) {
        ...
        if (CONDITION && CONDITION && CONDITION) {
            ...
            nextByte();
            if (CONDITION && CONDITION && CONDITION) {
                nb = (byte)(nb * 8 + currByte() - ox30); //check on this
            }
        }
    }
}
...
}
```

What are the effects??



Motivation

Java Source Compare

DSE/src/Logical1.java

```
4     int old;
5     int[] data;
6
7 public int logicalValue(int t){
8     if (!(currentTime - t >= 100)){
9         return old;
10    }else{
11        int val = 0;
12        for (int i=0; i<data.length; i++){
13            val = val + data[i];
14        }
15        old = val;
16        return val;
17    }
18}
19}
20}
```

DSE/src/Logical2.java

```
4     int old;
5     int[] data;
6
7 final int THRESHOLD = 100;
8 public int logicalValue(int t){
9     int elapsed = currentTime - t;
10    int val = 0;
11    if (elapsed < THRESHOLD){
12        val = old;
13    }else{
14        for (int i=0; i<data.length; i++){
15            val = val + data[i];
16        }
17        old = val;
18    }
19    return val;
20}
```

The screenshot shows a Java Source Compare tool interface. It displays two Java files side-by-side: DSE/src/Logical1.java and DSE/src/Logical2.java. The code in both files is identical, showing a method logicalValue that calculates a sum based on a threshold. Two specific sections of code are highlighted with red boxes: the inner loop in Logical1.java (lines 12-14) and the inner loop in Logical2.java (lines 14-16). Arrows from the highlighted code in Logical1.java point to the corresponding code in Logical2.java, suggesting a transformation or comparison between the two versions.

Differential Symbolic Execution

- Technique to detect and characterize the effects of program changes in terms of *behavioral* differences between program versions
 - Multi-stage analysis
 - Combines state-of-the-art symbolic execution techniques with over-approximating symbolic summaries

Differential Symbolic Execution

Java Source Compare

DSE/src/Logical1.java

```
4 int old;
5 int[] data;
6
7 public int logicalValue(int t){
8     if (!(currentTime - t >= 100)){
9         return old;
10    } else{
11        int val = 0;
12        for (int i=0; i<data.length; i++){
13            val = val + data[i];
14        }
15        old = val;
16        return val;
17    }
18}
19
20
```

DSE/src/Logical2.java

```
4 int old;
5 int[] data;
6
7 final int THRESHOLD = 100;
8 public int logicalValue(int t){
9     if (!(currentTime - t >= THRESHOLD)){
10        return old;
11    } else{
12        int val = 0;
13        for (int i=0; i<data.length; i++){
14            val = val + data[i];
15        }
16        old = val;
17    }
18    return val;
19}
```

No functional differences found.

OK

Validate Refactoring

Source Line Diff:
50+% of source lines changed

vs.

DSE:
no functional differences found

Overview of Presentation

- DSE methodology
- Summarizing program behavior
- Notions of equivalence and deltas
- Applications of DSE
- Conclusions and future work

Symbolic Execution

```
int computeFine(int balance){  
1: if (balance <= 0)  
2:     return 10;  
3: else  
4:     return 0;  
}
```

| | |
|-----------------------|--------------------|
| balance = -100 | RETURN = 10 |
| balance = -99 | RETURN = 10 |
| ... | |
| balance = 0 | RETURN = 10 |
| balance = 1 | RETURN = 0 |
| ... | |

VS.

| | |
|------------------------|--------------------|
| balance <= 0 | RETURN = 10 |
| !(balance <= 0) | RETURN = 0 |

Differential Symbolic Execution

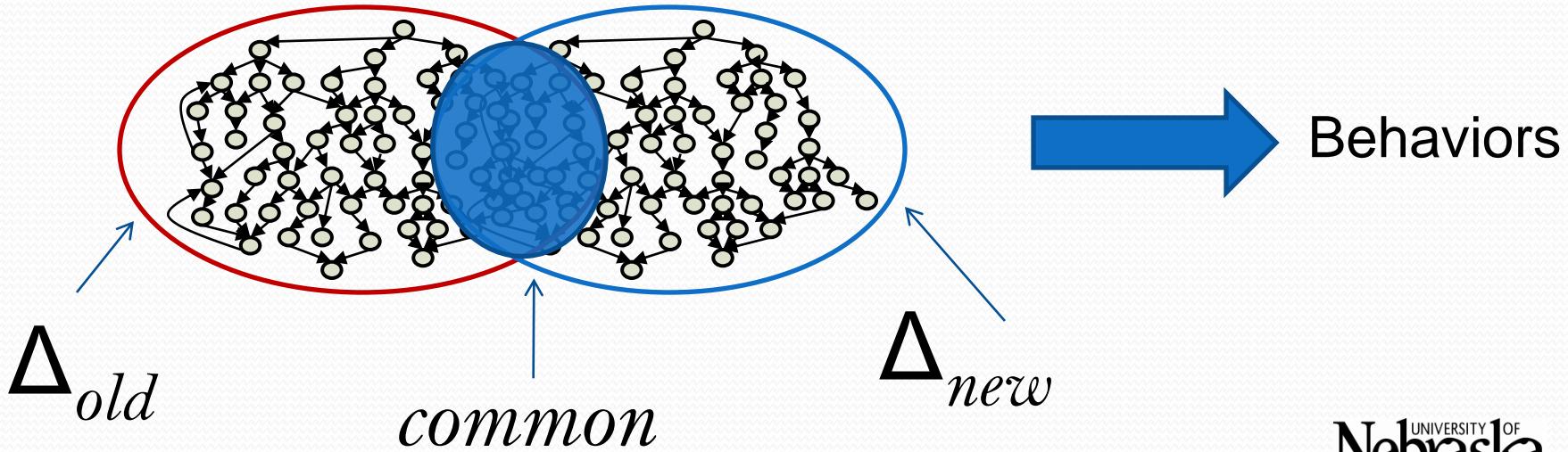
Java Source Compare

DSE/src/Logical.java DSE/src/Logical2.java

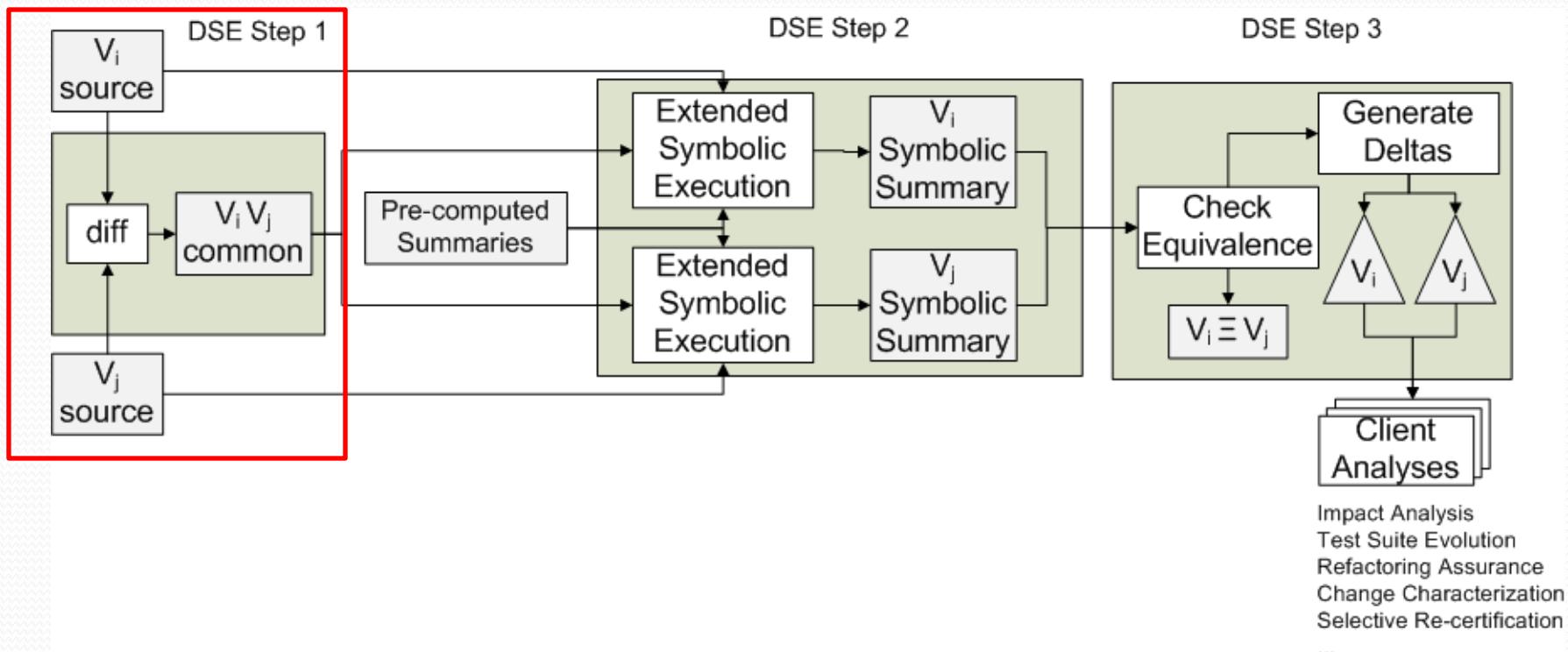
```
4 int old;
5 int[] data;
6
7 public int logicalValue(int t){
8     if (!(currentTime - t >= 100)){
9         return old;
10    }else{
11        int val = 0;
12        for (int i=0; i<data.length; i++){
13            val = val + data[i];
14        }
15        old = val;
16        return val;
17    }
18}
19}
20
```

```
4 int old;
5 int[] data;
6
7 final int THRESHOLD = 100;
8 public int logicalValue(int t){
9     int elapsed = currentTime - t;
10    int val = 0;
11    if (elapsed < THRESHOLD){
12        val = old;
13    }else{
14        for (int i=0; i<data.length; i++){
15            val = val + data[i];
16        }
17        old = val;
18    }
19    return val;
20}
```

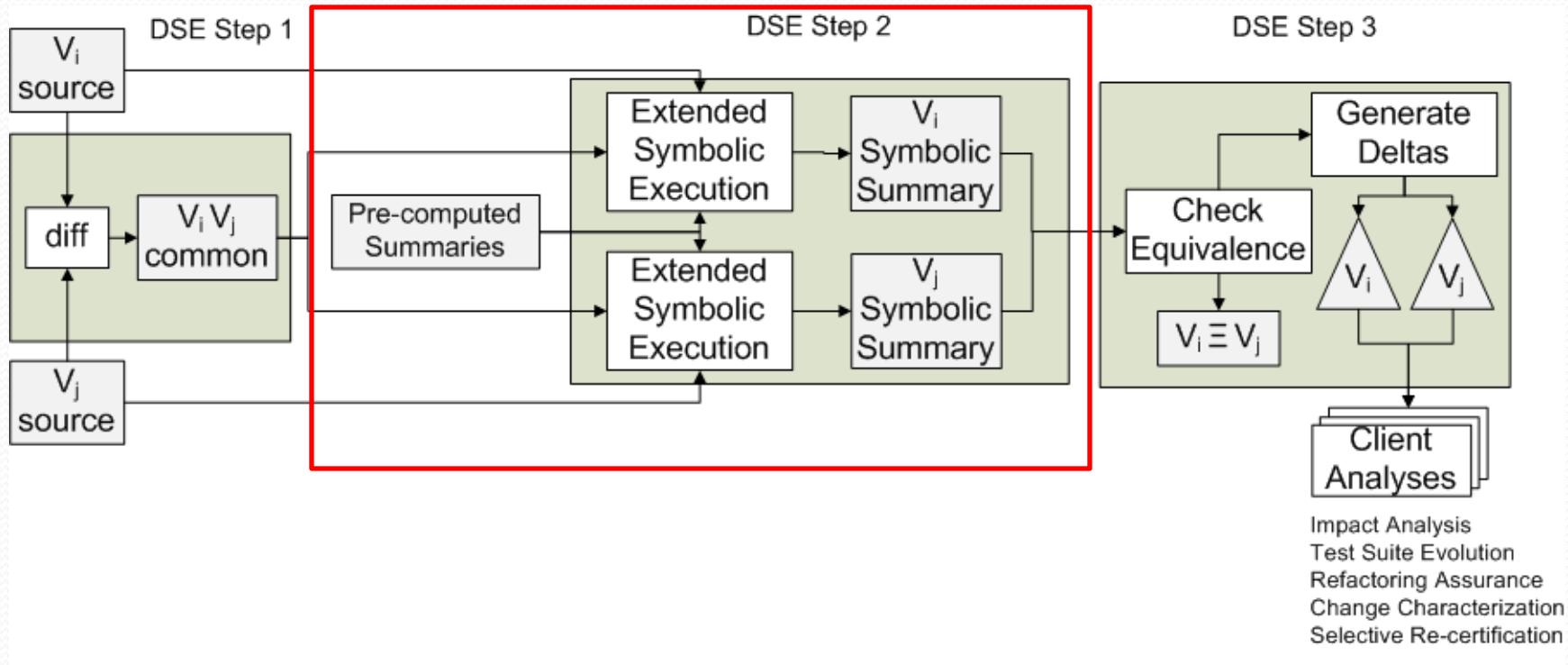
→ Locations



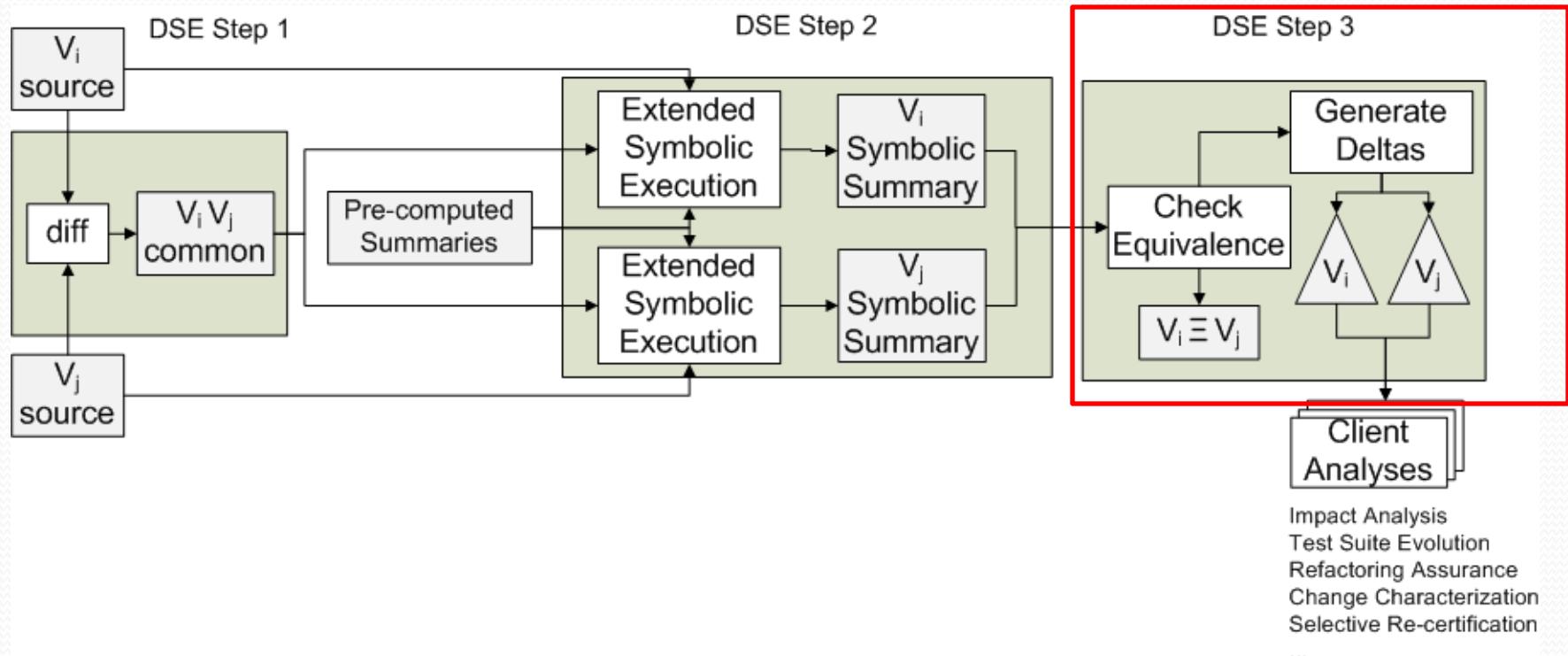
DSE Methodology



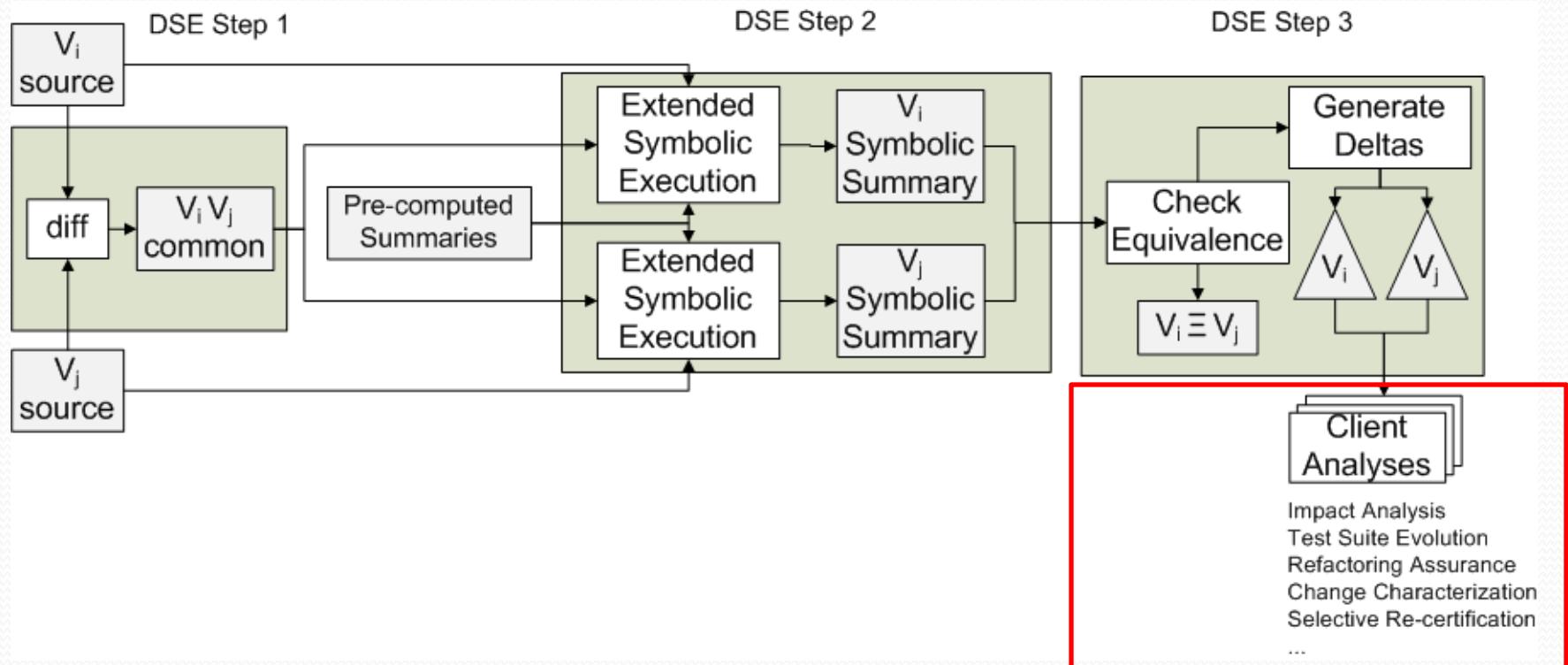
DSE Methodology



DSE Methodology



DSE Methodology



Overview of Presentation

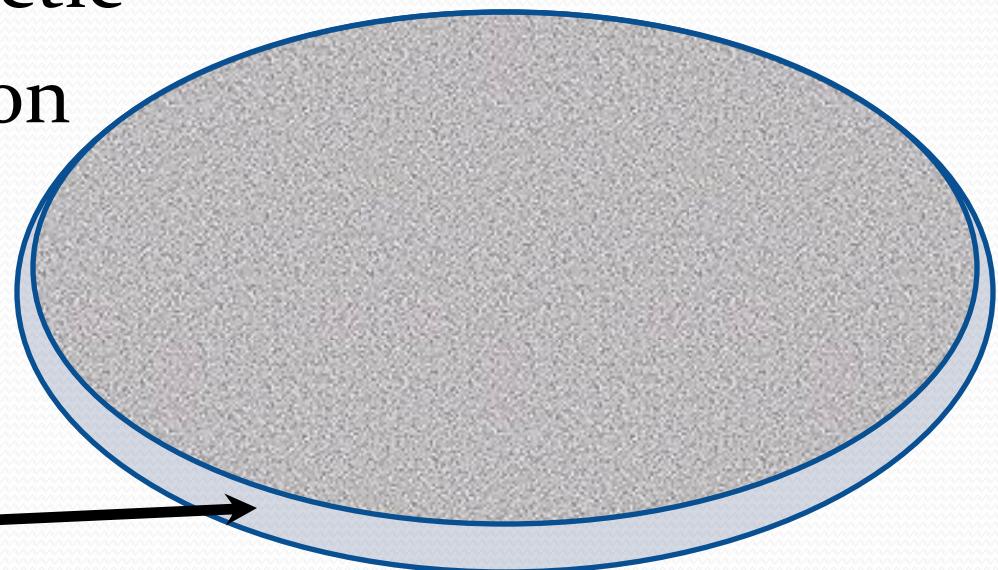
- DSE methodology
- Summarizing program behavior
- Notions of equivalence and deltas
- Applications of DSE
- Conclusions and future work

Summaries of Program Behaviors

- It is not always possible to compute complete summaries
 - Non-linear arithmetic
 - Loops and recursion

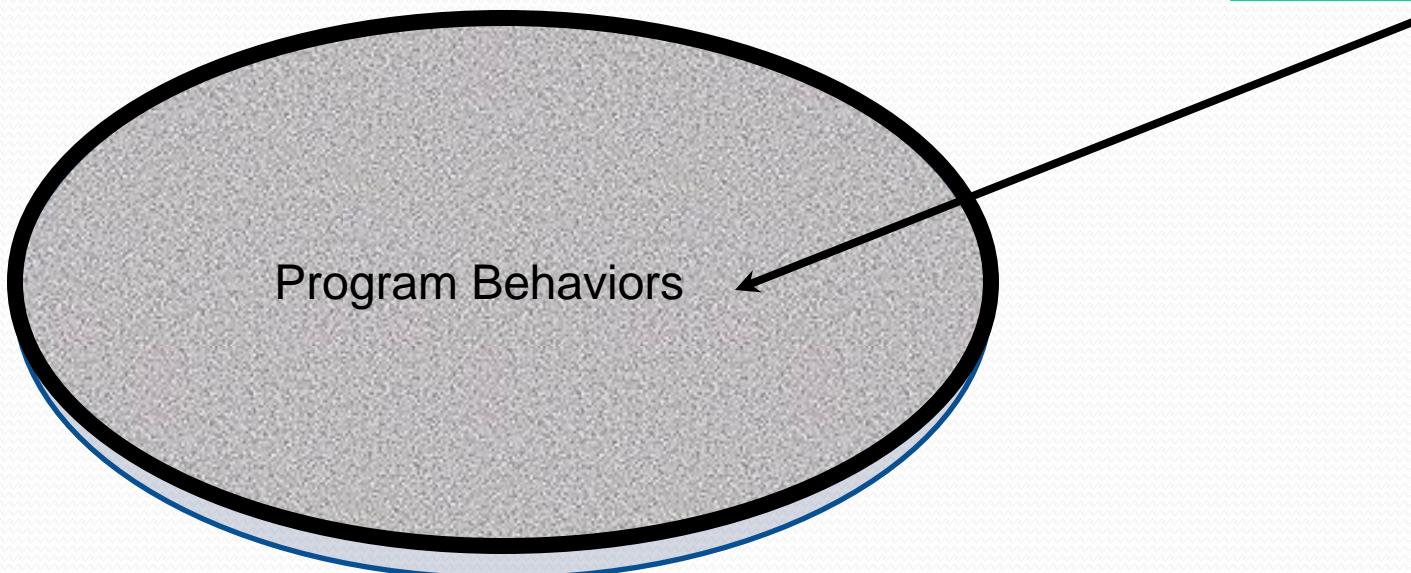


Incomplete



Incomplete Summaries

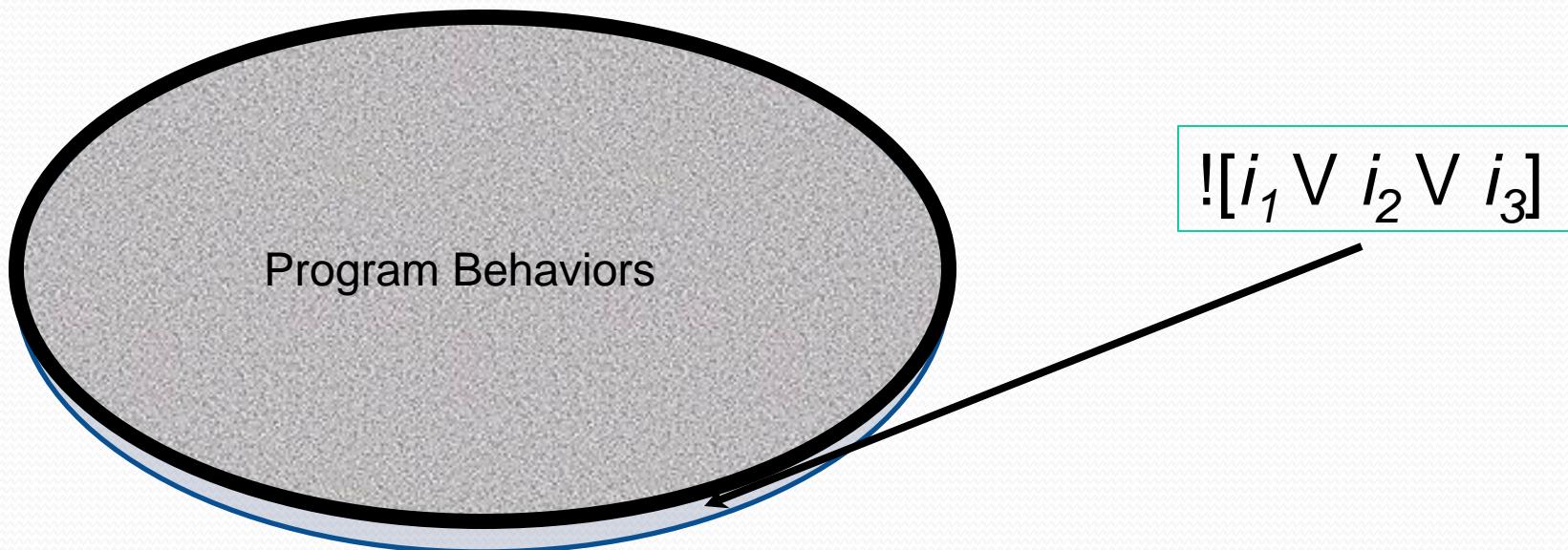
- Explicitly define the input space covered by the summary...



$$i_1 \vee i_2 \vee i_3$$

Incomplete Summaries

- ...then, focus subsequent analysis tool on behaviors not covered



Abstract Summaries on Common Code

Abstract Symbolic Summary:

Read:{x,tmp}
Write: {tmp,old}

```
void test(...){ // Vi
    S1;
    S2;
    S3;
```

```
//begin unchanged code
...
for (int i=0; i<len; i++){
    tmp = tmp + x[i];
}
old = tmp;
//end unchanged code
```

```
Sn;
Sn+1;
...
}
```

```
void test(...){ // Vj
    Sa;
    Sb;
    Sc;
```

```
//begin unchanged code
...
for (int i=0; i<len; i++){
    tmp = tmp + x[i];
}
old = tmp;
//end unchanged code
```

```
Sm;
Sm+1;
...
}
```

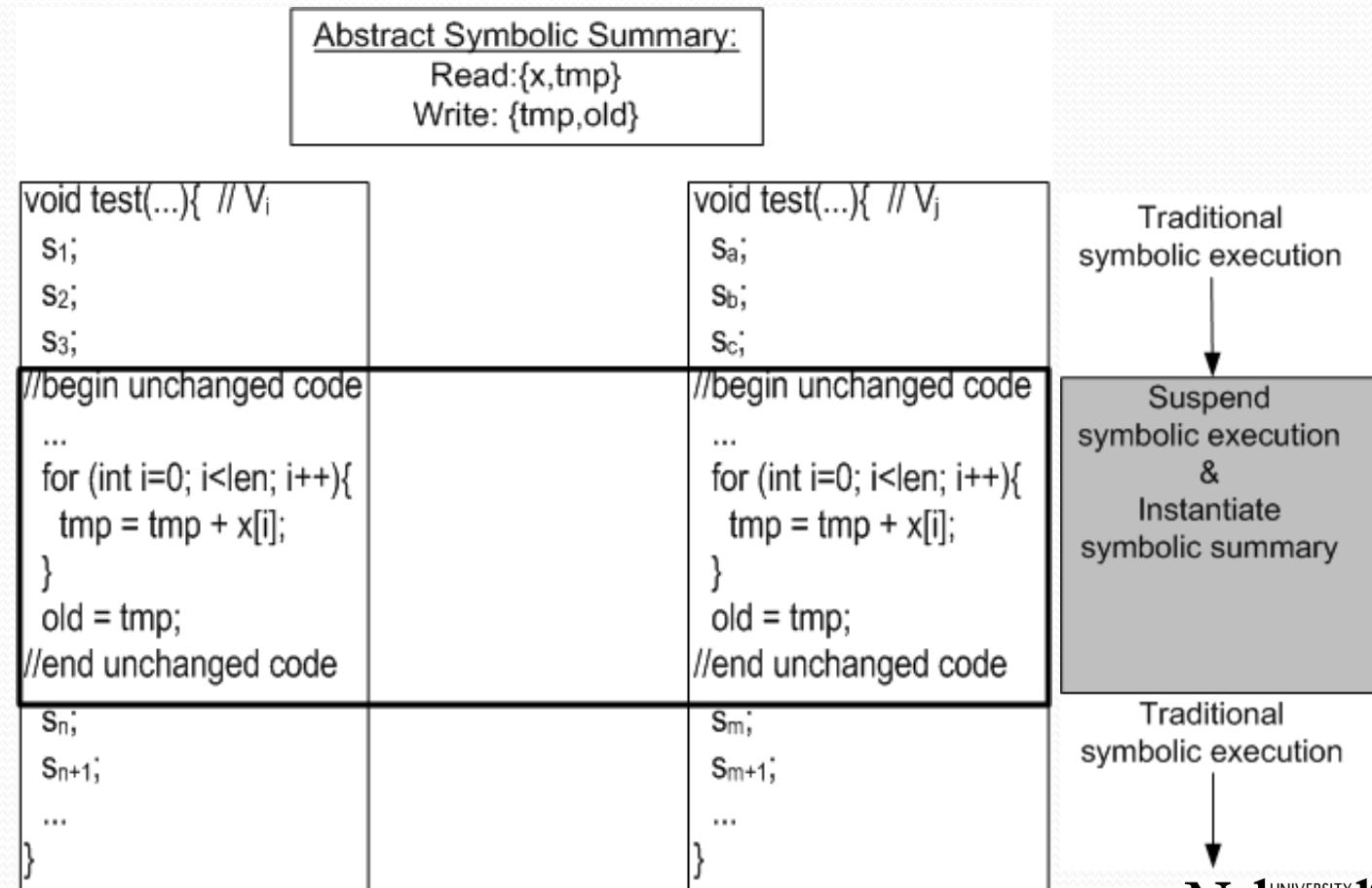
Abstract Summaries on Common Code

| | | <u>Abstract Symbolic Summary:</u> Read:{x,tmp} Write: {tmp,old} | |
|---|--|---|---|
| Traditional symbolic execution | | <pre>void test(...){ // V_i S₁; S₂; S₃;</pre> | <pre>void test(...){ // V_j S_a; S_b; S_c;</pre> |
| Suspend symbolic execution & Instantiate symbolic summary | | <pre>//begin unchanged code ... for (int i=0; i<len; i++){ tmp = tmp + x[i]; } old = tmp; //end unchanged code</pre> | <pre>//begin unchanged code ... for (int i=0; i<len; i++){ tmp = tmp + x[i]; } old = tmp; //end unchanged code</pre> |
| Traditional symbolic execution | | <pre>S_n; S_{n+1}; ... }</pre> | <pre>S_m; S_{m+1}; ... }</pre> |

Abstract Summaries on Common Code

| Abstract Symbolic Summary: | |
|---|---|
| Read:{x,tmp} Write: {tmp,old} | |
| void test(...){ // V _i S ₁ ; S ₂ ; S ₃ ; Update Path condition $\text{PC}_{v_i} : \dots \wedge \text{PC}_B(X_i, T_i)$ & Program state $v[\text{old}] : \text{old}_B(X_i, T_i)$ $v[\text{tmp}] : \text{tmp}_B(X_i, T_i)$ | //begin unchanged code ... for (int i=0; i<len; i++){ tmp = tmp + x[i]; } old = tmp; //end unchanged code S _n ; S _{n+1} ; ... } |
| | //begin unchanged code ... for (int i=0; i<len; i++){ tmp = tmp + x[i]; } old = tmp; //end unchanged code S _m ; S _{m+1} ; ... } |

Abstract Summaries on Common Code



Abstract Summaries on Common Code

| <u>Abstract Symbolic Summary:</u> Read:{x,tmp} Write: {tmp,old} | |
|---|---|
| <pre>void test(...){ // V_i S₁; S₂; S₃; //begin unchanged code ... for (int i=0; i<len; i++){ tmp = tmp + x[i]; } old = tmp; //end unchanged code S_n; S_{n+1}; ... }</pre> | <pre>void test(...){ // V_j S_a; S_b; S_c; //begin unchanged code ... for (int i=0; i<len; i++){ tmp = tmp + x[i]; } old = tmp; //end unchanged code S_m; S_{m+1}; ... }</pre> |
| | <p>Update Path condition $PC_{Vj} : \dots \wedge PC_B(X_j, T_j)$ & Program state $v[old] : old_B(X_j, T_j)$ $v[tmp] : tmp_B(X_j, T_j)$</p> |

Overview of Presentation

- DSE methodology
- Summarizing program behavior
- Notions of equivalence and deltas
- Applications of DSE
- Conclusions and future work

Checking Equivalence

- What does “equivalent” mean?
 - Refactoring assurance: are the externally observable behaviors the same?
 - Change characterization: given a set of inputs, how are the computed values affected?
- How do we describe the differences and common *behaviors*?

Checking Equivalence

```
//old  
int computeFine(int balance){  
1: if (balance < 0)  
2:   return 10;  
3: else if (balance == 0)  
4:   return 10;  
5: else  
6:   return 0;  
}
```

```
//new  
int computeFine(int balance){  
1: if (balance <= 0)  
2:   return 10;  
3: else  
4:   return 0;  
}
```

Checking Equivalence

| | | |
|-----------------------------|---|--------------------|
| $\text{old}_{\text{sum}} =$ | balance < 0 | RETURN = 10 |
| | balance == 0 | RETURN = 10 |
| | !(balance < 0) && !(balance == 0) | RETURN = 0 |

?
≡

| | | |
|-----------------------------|---------------------------|--------------------|
| $\text{new}_{\text{sum}} =$ | balance <= 0 | RETURN = 10 |
| | !(balance <= 0) | RETURN = 0 |

Functional Equivalence

$((\text{balance} < 0) \wedge \text{RETURN} = 10) \vee$

$((\text{balance} == 0) \wedge \text{RETURN} = 10) \vee$

$(\neg(\text{balance} < 0) \wedge \neg(\text{balance} == 0) \wedge \text{RETURN} = 0)$

?

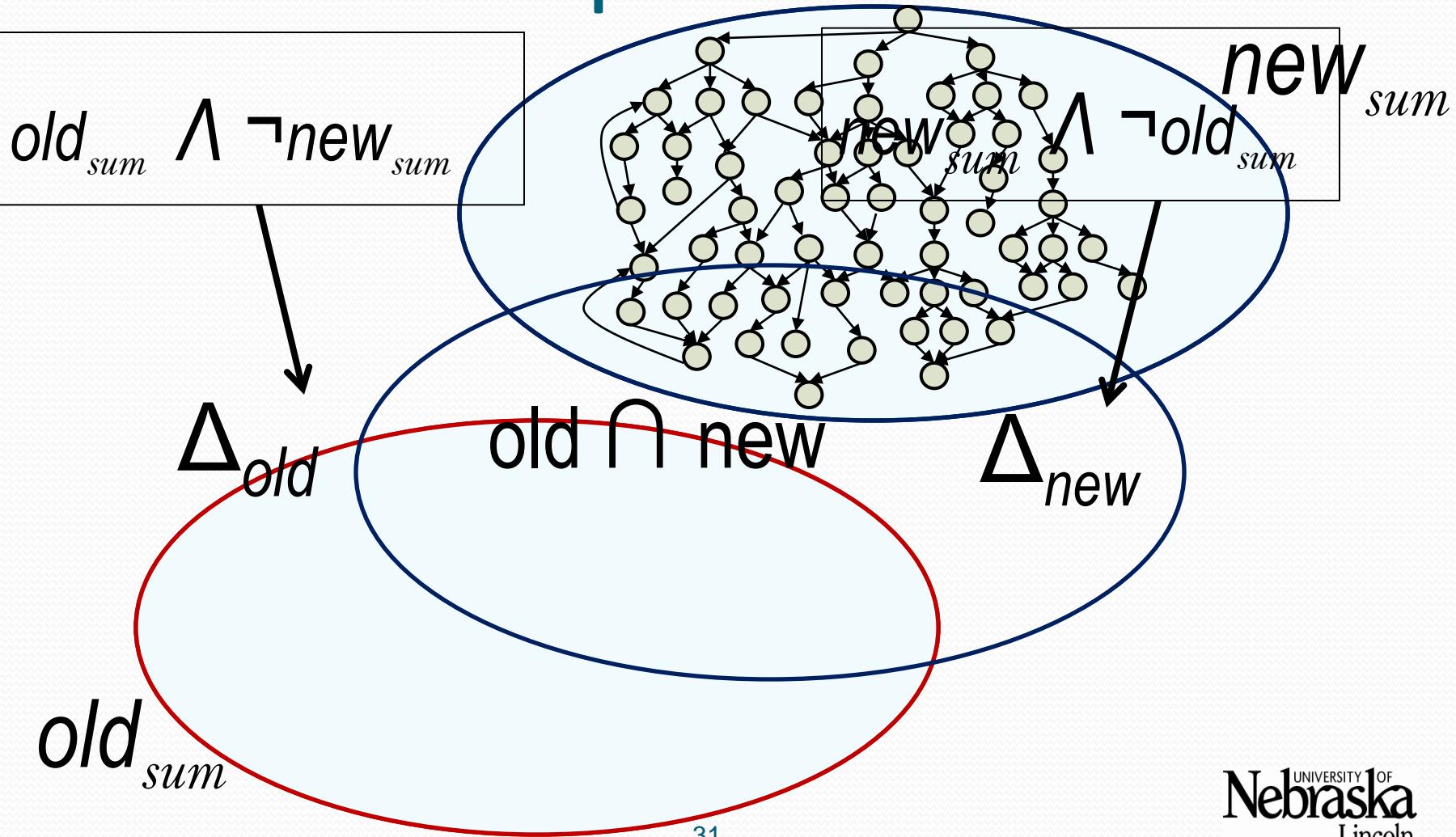
≡

$((\text{balance} \leq 0) \wedge \text{RETURN} = 10) \vee$

$(\neg(\text{balance} \leq 0) \wedge \text{RETURN} = 0)$

Functionally Equivalent? ✓

Functional Equivalence



Partition-effects Equivalence

((balance < 0) \wedge RETURN = 10) \vee

((balance == 0) \wedge RETURN = 10) \vee

(\neg (balance < 0) \wedge \neg (balance == 0) \wedge RETURN = 0)

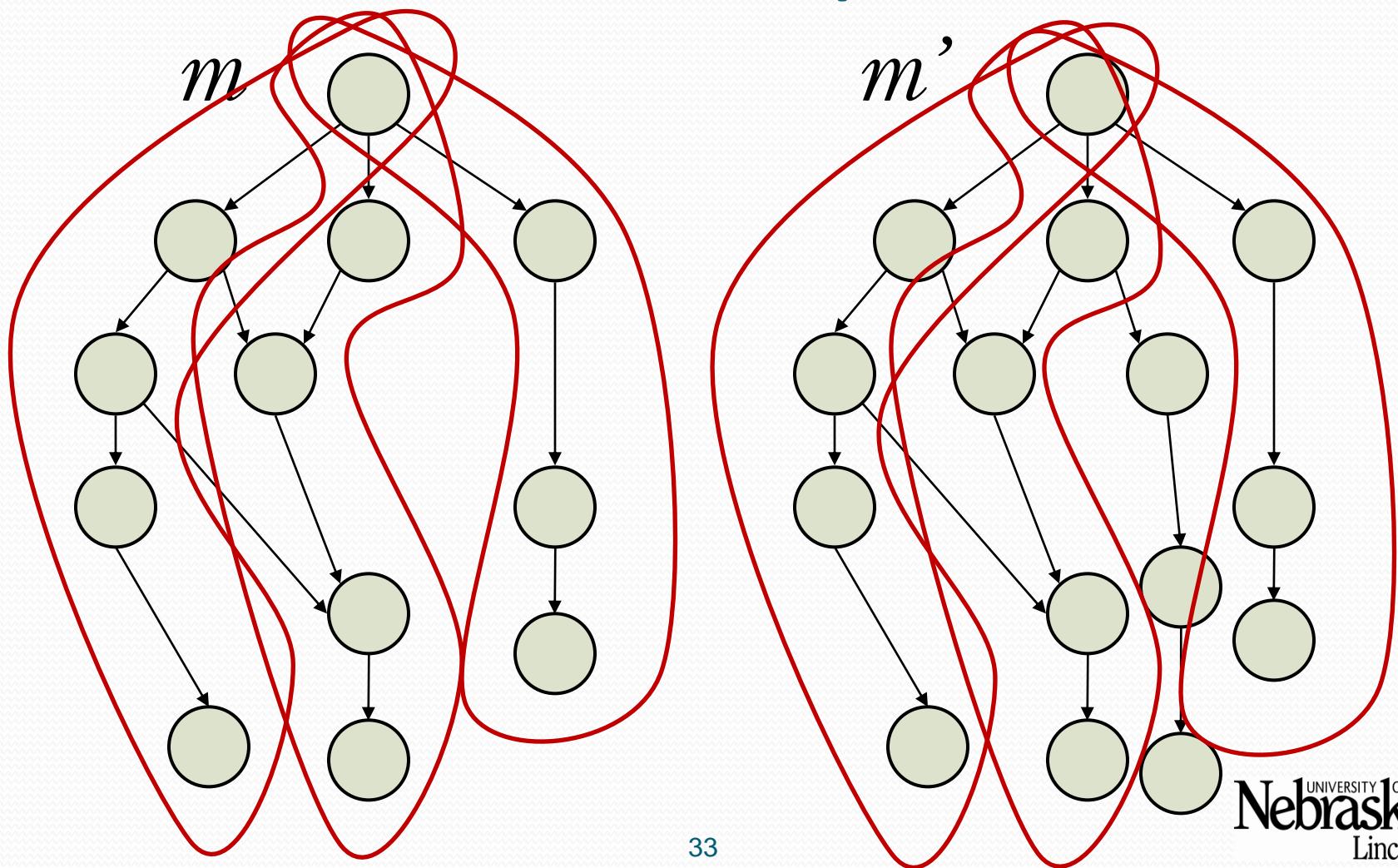
?
≡

((balance <= 0) \wedge RETURN = 10) \vee

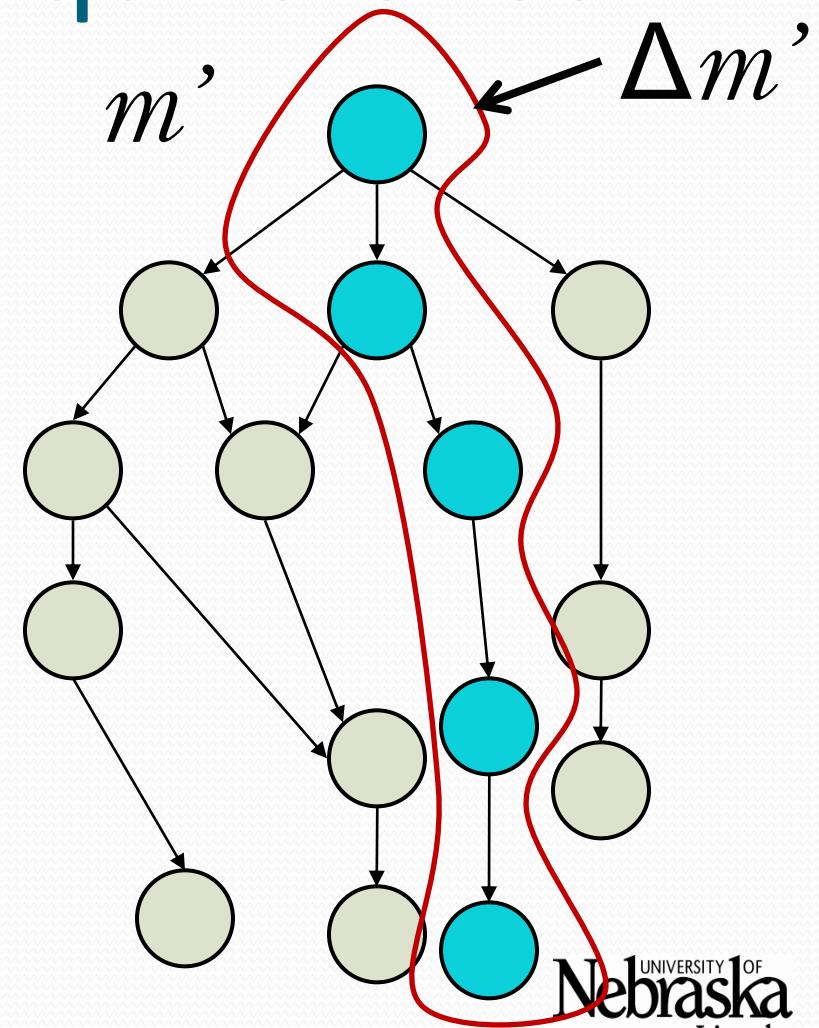
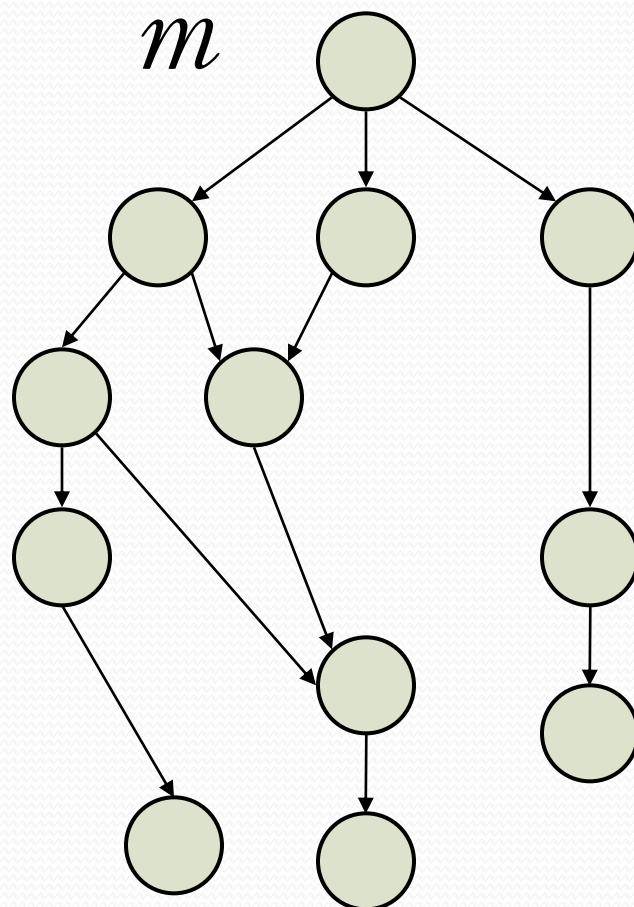
(\neg (balance <= 0) \wedge RETURN = 0)

Partition-effects Equivalent? X

Partition-effects Equivalence



Partition-effects Equivalence



Partition-effects Equivalence

| | | | |
|-----------------------|---|-----------------------|--------------------|
| δ_{old} | = | balance < 0 | RETURN = 10 |
| | | balance == 0 | RETURN = 10 |

| | | |
|----------|---|-------------------|
| Common = | !(balance < 0) && !(balance == 0) | RETURN = 0 |
|----------|---|-------------------|

| | | | |
|-----------------------|---|------------------------|--------------------|
| δ_{new} | = | balance <= 0 | RETURN = 10 |
|-----------------------|---|------------------------|--------------------|

Overview of Presentation

- DSE methodology
- Summarizing program behavior
- Notions of equivalence and deltas
- Applications of DSE
- Conclusions and future work

Applications of DSE

- Applied to artifacts from Software-artifact Infrastructure Repository (<http://sir.unl.edu>)
 - JMeter
 - Siena
- Client applications
 - Refactoring assurance
 - Test suite evolution
 - Change characterization

Change Characterization

```
//Siena version 3
public static boolean match(byte[] x, byte[] y){
    if (x.len != y.len) return false;
    for(int i=0; i<x.len; ++i)
        if (x[i] != y[i]) return false;
    return true;
}
```

```
//Siena version 4
public static boolean match(byte[] x, byte[] y){
    if (x == null && y == null) return true;
    if (x == null || y ==null || x.len != y.len) return false;
    for(int i=0; i<x.len; ++i)
        if (x[i] != y[i]) return false;
    return true;
}
```

match() Version 3

| Input Partition | Effect |
|---|---------------------------------------|
| X==null | RETURN == EXCEPTION |
| Y==null | RETURN == EXCEPTION |
| !(X==null) \wedge !(Y==null) \wedge (X.l != Y.l) | RETURN == FALSE |
| !(X==null) \wedge !(Y==null) \wedge (X.l != Y.l) \wedge PC _{B1} (T, X, Y) | RETURN == RET _{B1} (T, X, Y) |

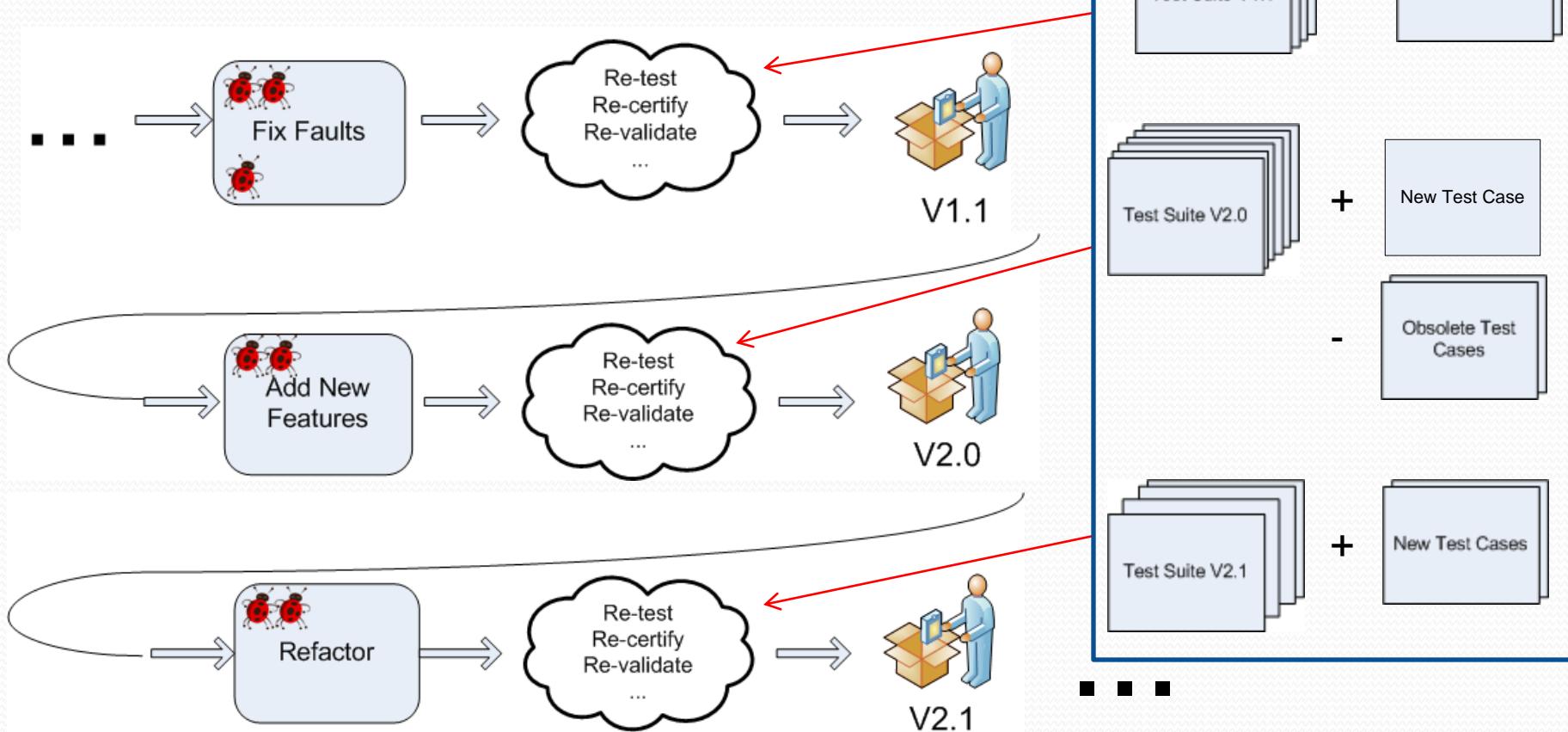
match() Version 4

| | |
|---|---------------------------------------|
| X==null \wedge Y==null | RETURN == TRUE |
| X==null \wedge !(Y==null) | RETURN == FALSE |
| !(X==null) \wedge Y==null | RETURN == FALSE |
| !(X==null) \wedge !(Y==null) \wedge (X.l != Y.l) | RETURN == FALSE |
| !(X==null) \wedge !(Y==null) \wedge (X.l != Y.l) \wedge PC _{B1} (T, X, Y) | RETURN == RET _{B1} (T, X, Y) |

Change Characterization

| On input | match() Version 3 | match() Version 4 |
|--|-------------------|-------------------|
| $x == \text{null} \wedge y == \text{null}$ | throws NPE | RETURN == TRUE |
| $x == \text{null} \wedge y != \text{null}$ | throws NPE | RETURN == FALSE |
| $x != \text{null} \wedge y == \text{null}$ | throws NPE | RETURN == FALSE |

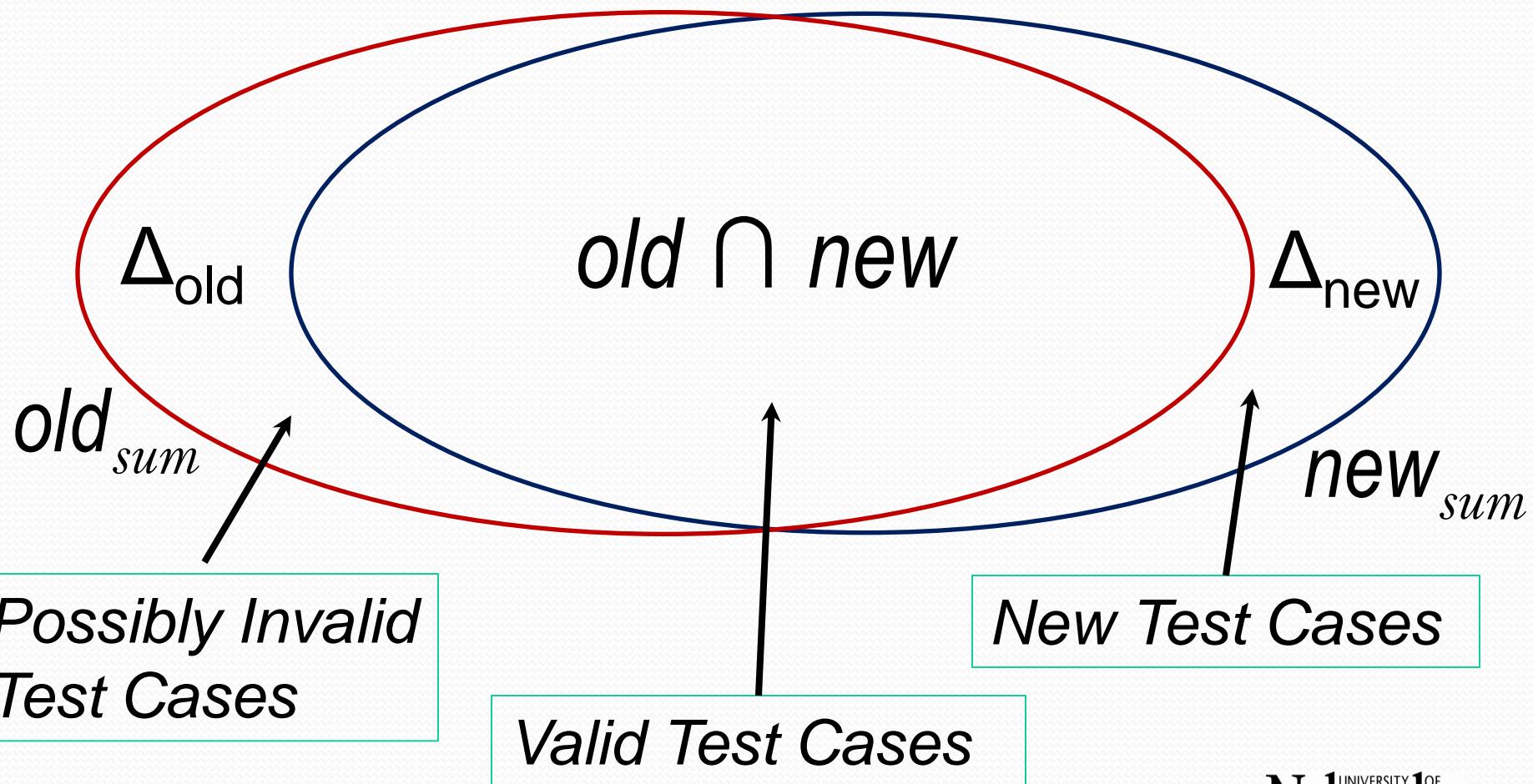
Test Suite Evolution



Automated Test Suite Evolution

- Automatically categorize existing test cases
 - Redundant test cases
 - Invalid (obsolete) test cases
- Provide developer hints to support test case re-use
- Identify missing test cases

Automated Test Suite Evolution



Overview of Presentation

- DSE methodology
- Summarizing program behavior
- Notions of equivalence and deltas
- Applications of DSE
- Conclusions and future work

Summary & Conclusions

- Contributions – Theoretical Foundations
 - Multi-stage, differential program analysis
 - Uses abstract symbolic summaries to leverage program commonalities
 - Enables client analyses to focus on the *behavioral differences*
 - Avoids fundamental limitations of symbolic execution

Summary & Conclusions

- Contributions – Theoretical Foundations
 - Two notions of equivalences and deltas
 - Functional
 - Partition-effects
 - Support for generalized symbolic execution of heap-manipulating Java programs*
 - Support for summaries of common code sequences*

*Tasks not listed in proposal

Summary & Conclusions

- Contributions – Implementation Framework
 - Text-based diff tool
 - AST-based diff tool
 - Extended symbolic execution (built on JPF & interfaces with CVC₃ theorem prover)
 - Equivalence checker (interfaces with CVC₃ theorem prover)

Summary & Conclusions

- Contributions – Practical Application
 - Change characterization
 - Refactoring assurance
 - Automated test suite evolution (ATSE)

Future Work

- Extend Differential Symbolic Execution
 - Apply DSE to other imperative models
 - Further explore abstract symbolic summaries
 - Partially interpreted uninterpreted functions
 - Multiple uninterpreted functions
 - Use only “as necessary”

Future Work

- Create hybrid analyses
 - Light-weight analyses
 - Impact analysis to drive symbolic execution into only the parts of the state space that are changed
 - Other techniques
 - Demand-driven symbolic execution

Future Work

- Explore new client applications of Differential Symbolic Execution
 - Software re-certification
 - Change characterization and metrics
 - Refactoring assurance
 - ...
- Perform empirical studies of cost and effectiveness of DSE

Differential Symbolic Execution

*Suzette Person
Dissertation Defense
July 8, 2009*

PhD Committee
Matthew Dwyer, Advisor
Myra Cohen
Sebastian Elbaum
David Rosenbaum



Related Work

- Currie et al. (Int'l Journal Par. Prog. '06)
- Bryant et al. (TCL '01)
- Jackson et al. (ICSM '94)
- Neamtiu et al. (MSR '05)
- Apiwattanapong et al. (ASE '07)
- Santelices et al., Apiwattanapong et al. (ASE '08, TAIC PART '06)
- Siegel et al. (ISSTA '06, PVM/MPI '08)